Asc 3

Machine introduction representation

BUS Interface unit – deals with address computation

HOW TO EXPRESS AN OFFSET

An assembler generates bytes

Proof: while you are in data segment you use data definition tools (db, dw, dd, dq). Why is this so important? When you are forced to declare variables in a separate data segment. What you have as a separated part of code will be generated separately by the compiler. The order in which you put variables is important because the order and size of the variables. The assembler is always capable of detecting which is the offset of the variable we declare.

THE offset part:

The machine instruction representation:

[prefixes] + code + [MODER/M] + [SIB] + [displacement] + [immediate]

MODER/M

Mod - 6-7

Reg/Opcode – 3-5

R/M – 0-2

* It has 256 possible values
* a byte which has has it’s structure segmented into 3 categories (r/m from a register or a memory operand)
* The byte rm decides wether it is a register or a memory operand
* If the field regopcode will select which is the register which will be the source/second operand
* If you have an operand from memory the first column will tell you which type the operand is
* If you want to know what the moder/m you need to understand the offset specification formula.

[] – dereferencing operator – an indication to the processor that you as a programmer need the value of the address, not the address itself

Offset specification formula

* Offset\_adress = [base] + [index \* scale] + [constant]
* [] – optionality as well
* Base
  + the name of the array (a7, a-base, 7-index)
  + it can be any general register of the 8 general registers that are specified in the executive unit
* Index
  + Any general register (EXCEPT ESP)
* Scale
  + Associated to the data types (db – 1, dw – 2, dd – 4, dq – 8)
  + Constant
* Constant
  + In this formula the constant is any constant value able to be determined/computed/identified at assembly time
* Every line in which you write a memory operand (NOT A VARIABLE)
* If we have at least one register,
* Mov eax, ebx
  + EAX = 00F748B9
* Mov eax, [ebx]
  + [ebx] will be used as an offset into the memory. The value will go into the memory at 00F748B9 and will transfer the value from there to eax (this is called indirection). Basically, [EBX] is a pointer value. It forces you to go into memory first and take the address from there
  + It obeys the offset specification formula, so it is right
  + For example, [v], [ebx] are both memory addressing operands. The register present inside the [] will act as a memory address.
* We are interested in understanding the mechanism which works the same in both case
* Mov eax [ecx + 2 \* edx – 7]
  + The contents of ecx are added to 2 \* the contents of edx and substract -7. You take 4 bytes from that address and put the bytes into eax. In memory everything is presented in linear order, but not in registers
* Mov eax 23
* Mov eax [23]
  + You go into offset 23 (next week we discuss in which segment) and take the value from there
  + When you reach this instruction you will see it disassembled. it will give you the representation in base 2. It becomes an error. Oly does not accept that you have a constant there
* Mov eax, [ecx + 2 \* esp + 7]
  + It will be a syntax error. Esp cannot be an index register
* Mov eax, [ebx\*2]
  + Ebx is an index register, so it works
* Mov eax, [ebx \* 3 ] <=> mov eax, [ebx + ebx\*2]
  + The assembler will always check if you put the stupidity that you write in a value
  + It works, for example, for 9
  + It doesn’t work for 7 because 6 cannot be used as a scale.
* For an instruction there are 3 ways to express a required operand
  + Register mode, if the required operand is a register (mov eax, 17)
    - In mov eax, ebx both the operands are used for registers, so you cannot use the formula.
  + Immediate mode
    - when we use directly the operand’s value (not in address and neither a register holding it) (mov eax, 17)
  + Memory adressing mode
* We used the formula from 2am (unconsciously) before when we used memory addressing operands. How does mov eax, [v] fit in our offset specification formula? The contents of a variable can never be a constant. on the other hand the value of a variable is not a constant determined at assembly time. The contents of the variable is the variable, but it’s address/offset of the variable is **always** a constant determinable at assembly time. This is why you see in oly “mov eax, [0042B7A9]”, the compiler doesn’t care what name you put, or jmp [0084] (below arrow). It can determine the offset which is a name at any time. The segment parts in the address specification formula are always used in pairs. The offset of a variable can always be determined at assembly time. This is why if you are using at least one register between square brackets it will be called INDIRECT ADRESSING. If you are using only the name of a variable, from the point of 2am formula, it will just be a constant. The + are not dealing with the contents of the variable, but represent the calculus of offset.
  + A particular case is specified when you only the name of the variable. It is called **direct addressing**.
* SIB
  + SCALE INDEX BASE
* THE SIB BYTE STRUCTURE
  + Scale – 6-7
    - 2 bits is enough for representing 4 values
      * 00 – scale is 1
      * 01 – scale is 2
      * 10 – scale is 4
      * 11 – scale is 8
  + Index – 3-5
  + Base – 0-2
* DISPLACEMENT
  + Mov eax, [eax+4\*ebx+7] - BOTH DISPLACEMENT AND IMMEDIATE
  + Mov eax, [v-7]
  + Mov eax, [v] – ONLY DISPLACEMENT
  + Mov eax, [7] – ONLY IMMEDIATE PART
  + Perfect for simple scaling variables
* If you can have immediate a register and an immediate
* If you have a register you stop, but if you have a memory variable you have a sib and maybe an immediate
* Mov [eax], ebx
  + It is ok, because the source operand will go into the memory and starting from that memory address you will put 4 bytes
* Mov [eax], [ebx]
  + Syntax error because only one may be stored in ram memory, the other one must be either a register or a constant
* Mov [eax], 23
  + Works, you can put a constant into memory, but it doesn’t work in how many bytes he puts 23. You don’t put 23 in eax, but in the memory eax points to
* Mov eip, eax
  + It will say “eip symbol not defined” it doesn’t allow you to use eip as a register, but as a variable
  + You can change the values in cs and eip using jumps

CODE TABLE

B8 - mov eax Iv (immediate value)

BA – mod edx Iv

50 - Push eax,

52 – push edx